

AD-A196 110

2
DTIC FILE COPY

OPLAN
CONSULTANT SYSTEM

Deliverable No.: B002

DTIC
SELECTED
S JUL 06 1988 D
G2 D

Prepared by:
JAYCOR

Prepared for:
Naval Research Laboratory
Washington, DC 20375-5000

In Response to:
Contract #N00014-86-C-2352

DISTRIBUTION STATEMENT A
Approved for public release
Distribution Unlimited

12 May 1988

INTRODUCTION

CKLOG (Calculus for Knowledge in LOGic) is a system created by C.V. Srinivasan. It is used for knowledge representation and problem solving in OPLAN-CONSULTANT, an expert system for Naval Operational Planning. The User Interface for the CKLOG System was created by Karen Weiss. The CKLOG System and The User Interface are both written in Common Lisp and run on the Symbolics Lisp Machine.

This document is divided into two parts: The CKLOG System and The User Interface. The CKLOG System section explains three major concepts in CKLOG - *contexts*, *event numbers*, and *classes*. With each concept description is an explanation as well as the data structure used in the code to represent the concept. The *classes* portion is further subdivided into parts explaining the notions of *dimensions*, *instances*, and *relations*.

The User Interface section is a manual on how to use the Lattice screen display and mouse commands to work with the data representations of CKLOG's major concepts. The first part describes the screen layout, explaining the purpose of the various panes in the window. The second and third parts describe how to manipulate *context* information and *event number* information, respectively. The fourth part describes how to manipulate *class* information. This portion is further subdivided into five parts. The first three parts explain the effects of clicking on each of the three mouse buttons. The fourth part explains how to use dimension defining facility. The fifth part explains how to use the class finding facility.

Accession For	
NTIS CRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By <i>per ltr.</i>	
Distribution /	
Availability Codes	
Dist	Avail and/or Special
A-1	



THE CKLOG SYSTEM

Contexts

All knowledge in the CKLOG system is defined in a *context*. A context describes the environment in which the knowledge is represented. The data structure that contains this environmental information is defined in the code as CNTXTN. The fields of the data structure and their initial values are as follows:

(p NIL)	;parent context
(ls NIL)	;left son context
(rs NIL)	;right son context
(seq NIL)	;sequent, or name of context
(subs NIL)	;substitutions used in problem solving
(offsp NIL)	;offspring used in problem solving
(tr NIL)	;true residues in context
(?r NIL)	;unknown residues in context
(fr NIL)	;false residues in context
(enums NIL)	;event numbers defined in context
(open T)	;flag telling whether or not goal is still open
(vars (LIST EN--HEIGHT!))	;variables defined in context
(constants (LIST EN--HEIGHT!))	;constants defined in context
(classes (LIST EN--HEIGHT!))	;classes defined in context
(undone* (LIST NIL))	;field currently undefined
(updates* (LIST NIL))	;list of updates
(history* (LIST NIL))	;list of commands
(future* (LIST NIL))	;list of undone commands
(+times (LIST NIL))	;plus time expressions in context
(etimes (LIST NIL))	;absolute times defined in context
(-times (LIST NIL))	;minus time expressions in context
(pad (MAKE-HASH-TABLE :TEST #'EQUAL :SIZE 128))	;scratch pad hash table
(ws (MAKE-HASH-TABLE :TEST #'EQUAL :SIZE 512)))	;world state hash table

There can be multiple contexts at once. With multiple contexts, CKLOG can search different worlds to try to solve a problem. These contexts can be created for new searches. CKLOG can test out a solution to a problem by creating offspring worlds, and then trace back to the original environment without disturbing the current state of affairs.

The Context Lattice represents the relationship between these environments. They are linked together as a binary tree. The *parent* field of a context indicates which context is the root of a given context, and the *left-son* and *right-son* fields point to the context's descendants.

Event Numbers

All knowledge in the CKLOG system is defined with an *event number* associated with it. This event number can have an absolute time or can be relative to another event. The data structure that contains this temporal information is defined in the code as ENUM-DESN. The fields of the data structure are as follows:

(en NIL)	;name of event number (enum)
(before NIL)	;event numbers before enum
(after NIL)	;event numbers after enum
(same NIL)	;event numbers at the same time as enum
(jbefore NIL)	;event number just before enum
(jafter NIL)	;event number just after enum
(context CONTEXT!)	;context in which enum is defined
(events NIL)	;events associated with enum
(time NIL)	;absolute time associated with enum
(retime NIL)	;relative time associated with enum
(condns NIL))	;conditions associated with enum

Absolute event times specify the year, month, day, hour, minute, and second of an event. This can be used as a time stamp for an event taking place at a certain date, or as a time value to be added or subtracted from an event date. An example of this would be in describing an event due to occur two weeks after another event.

Relative event times are represented by *time expressions*. These can describe simple ordering relationships, such as (after En0), (before En1), (justafter En2), and (justbefore En3). When a time expression specifies an *after* relationship, as in (after En0), it means that there exists a point in time, t, such that t is after En0. Other events may occur between these two points. *Before* relationships are similar. When a time expression specifies a *justafter* relationship, as in (justafter En2), it means that there exists a point in time, t, such that t is justafter En2. No other events may occur between these two points. If t is justafter En2, then En2 is justbefore t.

Time expressions can also describe an interval, such as (between En0 En1) and (during En2 En3). The expression (between En0 En1) means that there exist three points in time, En0, En1, and t, such that t is after En0 and before En1. Other events may occur between En0 and En1. The expression (during En2 En3) specifies all points starting at En2 and up to but not including En3.

Time expressions can describe an equation involving two event numbers, such as (plus En0 En1) and (minus En3 En2). The event numbers can be absolute or relative. If both are absolute, then CKLOG solves the equation and create a new event number occurring at the solution time. If one or neither is absolute, then the equation isn't solvable until more information is acquired, and the associated event is stored in the time lattice according to what information is known. This information includes the fact that all event numbers are after En0, the beginning of time for CKLOG, and before En10000000, CKLOG's infinity. In the time expression (plus En0 En1), the associated event number is located in the lattice after both En0 and En1. In the time expression (minus En3 En2), all that is known is that the associated event number is located in the lattice before En3.

Classes

As previously stated, a *context* is the world in which information exists. An *event number* is the time at which a piece of information was entered into the world. A *class* is the type of object which the information describes. The data structure that contains this object information is defined in the code as CLPROPS. The fields of the data structure are as follows:

(gens (LIST EN--HEIGHT!))	:generalizations of the class
(spes (LIST EN--HEIGHT!))	:specializations of the class
(ins (LIST NIL))	:instances of the class
(subcl (LIST NIL))	:enum associated w/creation of a subclass of the class
(enum (LIST ENUM!))	:event number associated with creation of the class
(cl NIL))	:name of the class

Classes are defined hierarchically, from the most general to the most specific. An example of this is the general notion of a physical object, then a more particular notion of a region, then more specifically a water region, which can be further refined to a lake, and finally reduced to a cubic volume. These refinements are called *specializations* of the general concept, which in turn is called a *generalization* of the smaller concepts. A class can be both a specialization and a generalization, such as **water-region** - a specialization of **region** and a generalization of **lake**. A class can have more than one specialization - **water-region** and **air-region** can also be specializations of **region**. A class can have more than one generalization - both **water-region** and **air-region** can be generalizations of **cubic-volume**.

When a class is a specialization of another class, such as **region** is a specialization of **physical-object**, and when that class has a specialization of its own, such as **water-region** is a specialization of **region**, then **water-region** is also a specialization of **physical-object**. Likewise, both **region** and **physical-object** are generalizations of **water-region**.

Dimensions

Once the types objects in the world are described, information can be created about the different kinds of objects. For example, a region has a location. In CKLOG, this would be stated as **(has-location region location)**. The converse of this can also be declared as **(location-of location region)**. These two statements are called *dimensions*. A dimension can be thought of as a basic sentence with a subject, verb, and object. A dimension belongs to the subject class, so in the above example, **(has-location region location)** is a dimension of the class **region**, and **(location-of location region)** is a dimension of the class **location**.

A dimension also has an upper bound and a lower bound associated with it. The upper bound of the dimension **(has-location region location)** is 1, since a region can have only one location. The lower bound is also 1, since a region must have a location. For the converse dimension, **(location-of location region)**, the upper bound is unlimited, since a location may be associated with many kinds of objects defined in the world. The lower bound is 0, since a location may be undefined for all the objects in the world.

When a dimension is defined for a class, all the specialization of that class inherit the dimension. Using the same example as above, **water-region** inherits the **(has-location region location)** dimension.

A dimension can also have *flags* which provide further information about the way the subject and object relate to each other when they are of the same class. The flags are symmetric, reflexive, irreflexive, and transitive. A dimension with a symmetric flag has the quality that the subject and object can be interchanged. An example of this is (**adjacent-to** **region** **region**), since regions are adjacent to each other. When a dimension is symmetric, the converse dimension is the same as the original.

In a dimension with a reflexive flag, the subject and object can denote the same entity. An example of this is (**supports** **force** **force**), since in a military situation a force would certainly support itself. In a dimension with an irreflexive flag, the subject and object cannot denote the same entity. An example of this is (**is-within** **region** **region**), since a region cannot be within itself.

A transitive flag follows the rule that if $A = B$, and $B = C$, then $A = C$. This is best shown by example: (**is-within** **region** **region**). If **regionA** is within **regionB**, and **regionB** is within **regionC**, then **regionA** is within **regionC**.

Instances

An *instance* of a class is the attaching of an actual object to its type. An example of this is the instantiation of **water-region** with **Pacific-Ocean**. **Pacific-Ocean** is also an instance of the class **region**, since **region** inherits whatever properties are associated with **water-region**. To clarify the situation, **Pacific-Ocean** is called an *immediate instance* of **water-region**. There may be many instantiations of a single class, such as the further instantiation of **water-region** with **San-Francisco-Bay**.

Relations

Just as an instance is a fleshing out of a class skeleton, a *relation* is a fleshing out of a dimension structure. A relation takes a dimension and plugs instances into the class slots. Naturally, the instances must be of the appropriate class. The instantiation of a dimension into a relation can be seen in the following: (**is-within** **Pacific-Ocean** **San-Francisco-Bay**).

THE USER INTERFACE

Screen Layout

The interface for the CKLOG system is brought to the screen by typing the keys **SELECT-A**. The screen is entitled LATTICE, since its primary function is to display the three types of lattices of information in the CKLOG system: *Context*, *Time*, and *Class*. The main portion of the screen is occupied by the display area. (Figure 1)

File Info

On the left side of the screen are various information panes. The top pane is labeled *File info*. It displays prompts for two file pathnames: a loading file and a saving file. After the prompts appear default pathnames for the files. If the user wants to load a pre-written file of commands with which to build a domain, he moves the mouse over the default file name and clicks. If the left mouse button is clicked, the whole default pathname is erased and the prompt awaits a full pathname. If the middle mouse button is clicked, the default pathname becomes editable, and the user can change it using standard EMACS commands. These two options are documented on the black bar at the very bottom of the screen. When the user moves the mouse over almost any item on the screen, documentation appears in this area.

After specifying the file pathname, the user must type the **END** key to signal the end of input. At this point, the user must click the mouse on the next line of text which says *Click here to load* in order for anything to happen. If this is not done, no file is loaded even though its pathname has been specified. The user can load the file at any point in the interaction with the CKLOG system.

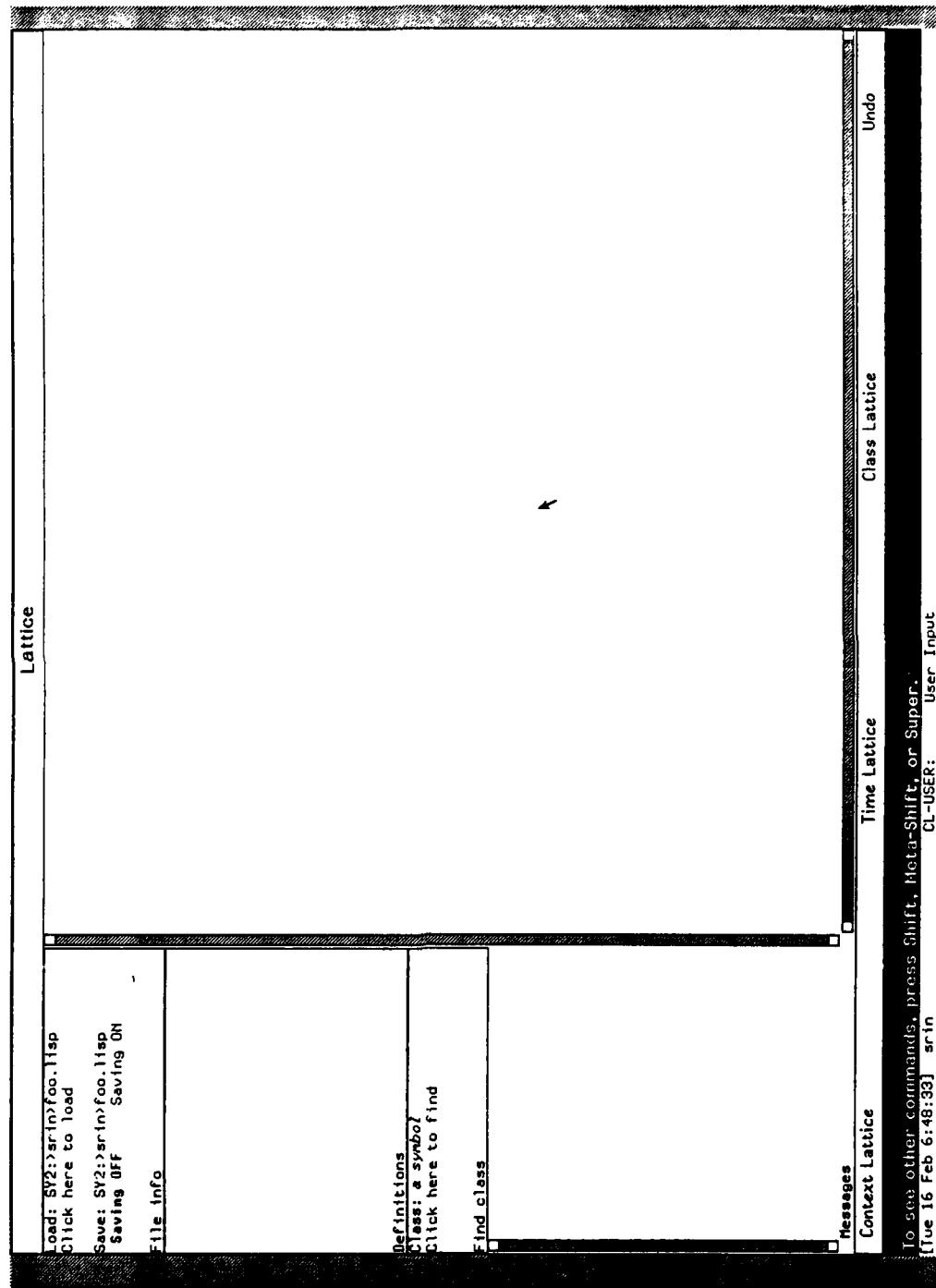
The other file option is to save domain-building information to a specified file. This means that any commands which build the user's world that are defined during the interaction can be saved to a given file. This file may then be loaded during another session, and the user can resume interaction with an identical world. Specifying the file pathname is the same as with the loaded file. The **END** key must be typed to signal the end of input. The user must then click the mouse over the words *Saving ON* in order to start recording commands to the specified file. When this is done, these words turn to bold and the words *Saving OFF* are no longer in bold. The default is to have no saving to a file. At any point in the interaction, the user can turn saving on or off. This is useful when testing out temporary commands.

Definitions

This pane, located under the *File info* pane, has different uses depending on what kind of lattice is being displayed. It stays blank for the Context Lattice, but is used for both the Time Lattice and the Class Lattice. Its uses are described later in the sections on the particular lattices.

Find Class

The next pane under the *Definitions* pane is used only for the Class Lattice, and is discussed in the Class Lattice section.



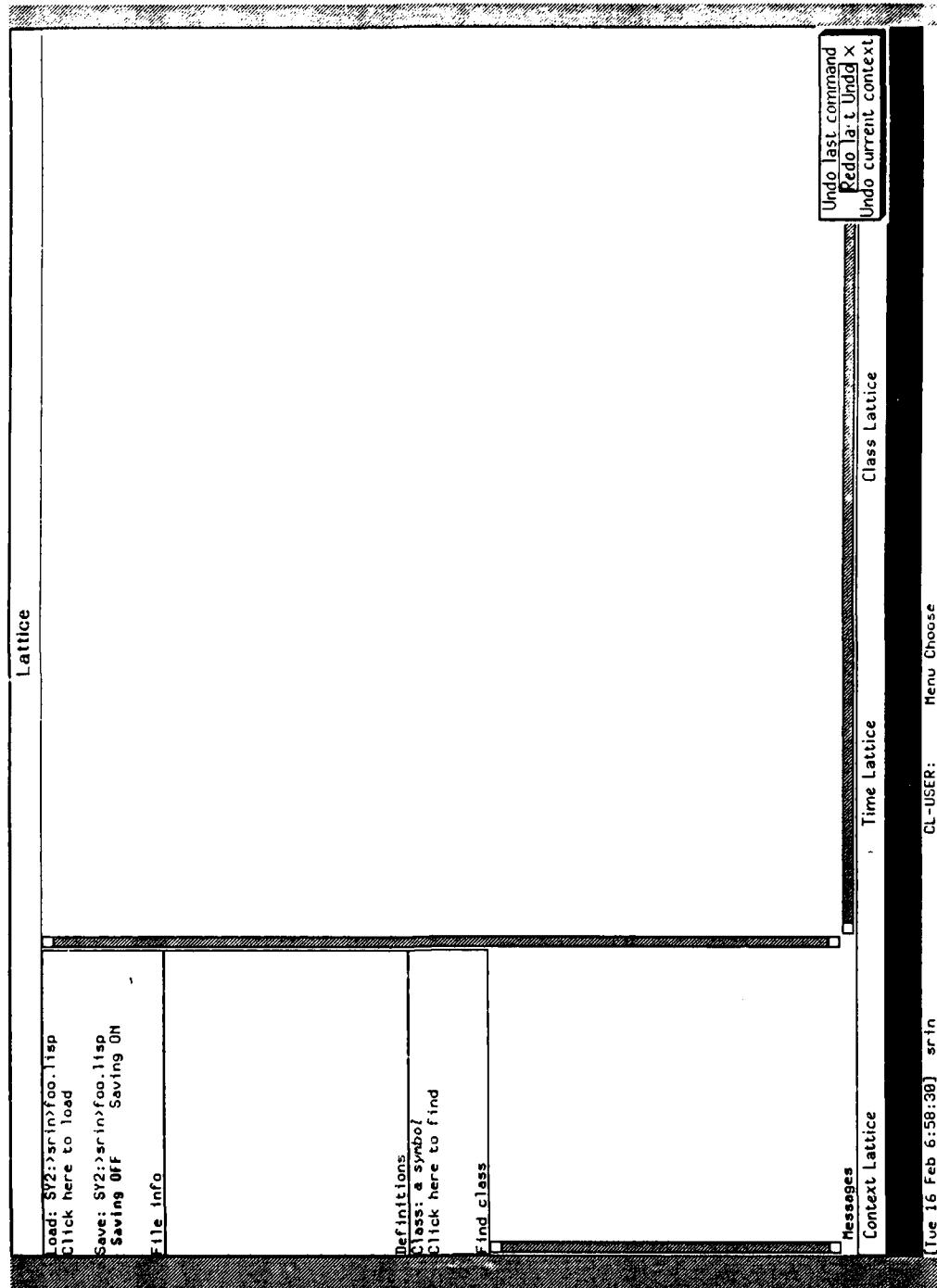
Messages

The bottom pane on the left side of the screen is used whenever the CKLOG system needs to print out a message. All commands send messages reporting the results of their execution. Since this pane is fairly small, messages often have unusual line breaks. The pane is scrollable, so all messages written to it during a session can be viewed at any time by moving the mouse over the gray bar on the left side and clicking the appropriate button. When the mouse is located any place along the scroll bar, the black documentation bar at the bottom of the screen instructs the user on how to click.

Command Menu

The user has the choice of viewing three different lattices: the *Context Lattice*, the *Time Lattice*, and the *Class Lattice*. Selection of a lattice is done by moving the mouse to the appropriate command in the menu line at the bottom of the screen, and then clicking on the command with the left mouse button.

Also included in this list is the *Undo* command. When clicked on, a menu pops up at the location of the mouse offering three choices: *Undo last command*, *Redo last Undo*, and *Undo current context*. (Figure 2) *Undo last command* removes any effects of the last command that the user evoked. If saving is on, it does not remove the last command from the file. This must be done manually. *Redo last Undo* reinstates the removed information. It also does not affect the saving file. *Undo current context* removes all information that has been defined in the current context. As with the other Undo commands, there is no effect on the saving file.



Context Lattice

When the user clicks on the *Context Lattice* command, the contexts names are drawn on the display pane with lines showing their relationships to each other. The current context's name is shown in bold. (Figure 3) Placing the mouse over a context name causes appropriate documentation to appear at the bottom of the screen. Clicking on the left mouse button causes the moused context to become the current context, if it isn't already. The display of its name changes to bold.

Clicking on the middle mouse button causes a window to pop up at the top of the screen, displaying all the values of the fields of the moused context. (Figure 4) To remove this window, the user must click anywhere outside of it.

Clicking on the right mouse button makes a new context after the moused context. The moused context is first changed to the current context if it isn't already, and then a new context is created after it. The display is redrawn to reflect this change. Since the context lattice is a binary tree, only two contexts can be created after a given context, occupying the fields of left-son and right-son. If the user attempts to add a third descendant context, an error message is printed on the message pane.

Time Lattice

When the user clicks on the *Time Lattice* command, the names of all defined event numbers are drawn on the display pane with lines showing their relationships to each other. (Figure 5) A single line represents a *before-after* connection, while a double line represents a *justbefore-justafter* connection. Placing the mouse over an event number causes appropriate documentation to appear at the bottom of the screen. The only action that can be taken with the mouse is to click on the middle button. This causes a window to pop up at the top of the screen, displaying all the values of the fields of the moused event number. (Figure 6) To remove this window, the user must click anywhere outside of it.

When the Time Lattice is displayed, the Definitions pane offers a prompt to enter an expression. Upon clicking on the prompt, the user can type a time expression enclosed in parentheses, and then click on the words *Define* or *Abort*. (Figure 7) *Define* incorporates the new event number into the Time Lattice and redraws the display to reflect this change. *Abort* wipes out the user's entry, causing no change to be made to the lattice.

Load: SV2:>sr in>foo.lisp
Click here to load

Save: SV2:>sr in>foo.lisp
Saving OFF Saving ON
File info

CONTEXT-1

CONTEXT-2

Lattice

Definitions
Class: a symbol?
Click here to find

Find class

((at Enl) ((T units-of SCALAR M ERIC-UNITS) (NIL 0)))
((at Enl) ((T is-a-unit METRIC-UNITS SCALAR) (NIL 0)))

Accepted.
((at Enl) ((T lowerbound-of DISTANCE-BOUNDS DISTANCE) (1 1)))
((at Enl) ((T is-lowerbound DISTANCE DISTANCE-BOUNDS) (NIL 0)))

Accepted.
((at Enl) ((T upperbound-of DISTANCE-BOUNDS DISTANCE) (1 1)))
((at Enl) ((T is-upperbound DISTANCE DISTANCE-BOUNDS) (NIL 0)))

Messages

Context Lattice

Mouse-L -R: Context Lattice.
To see other commands, press Shift, Meta-Shift, or Super.
[Tue 16 Feb 7:07:38] sr in Cl-USER: User Input

Time Lattice

Class Lattice

Undo

Load: SY2:>srin>foo.lisp
Click here to load

Save: SY2:>srin>foo.lisp
Saving OFF

File Info

Click here for time expression

Lattice

Definitions

Class: & symbol
Click here to find

Find class

Time lattice

Context Lattice

House-L, -R: Time Lattice, press Shift, Meta-Shift, or Super.

To see other commands, press Shift, Meta-Shift, or Super.

[Tue 16 Feb 7:18:51] srin

CL-USER: User Input

Messages

Class Lattice

Undo

En1000000000

En3

En4

En2

En1

En0

Diagram: A diamond-shaped lattice node with four edges. The top edge is labeled 'En3', the bottom edge is labeled 'En4', the left edge is labeled 'En1000000000', and the right edge is labeled 'En2'. Below the diamond, there are two horizontal lines: one ending at 'En1' and another ending at 'En0'.

Accepted.
 ((at En1) ((1 upperbound-of DISTANCE-BOUNDS DISTANCE (1 1)))
 ((at En1) ((1 is-upperbound DISTANCE DISTANCE-BOUNDS) (NIL 0)))
 (at (justafter En1) En2), Accepted
 (at (after En2) En3), Accepted
 (at (after En2) En4), Accepted
 (at (before En1000000000) En5), Accepted
 Accepted.
 (SETECOUNT
 (1before| 1En1000000001)
 UPDS!
 T) ... Undone.

CL-USER: User Input

Messages

Class Lattice

Undo

En1000000000

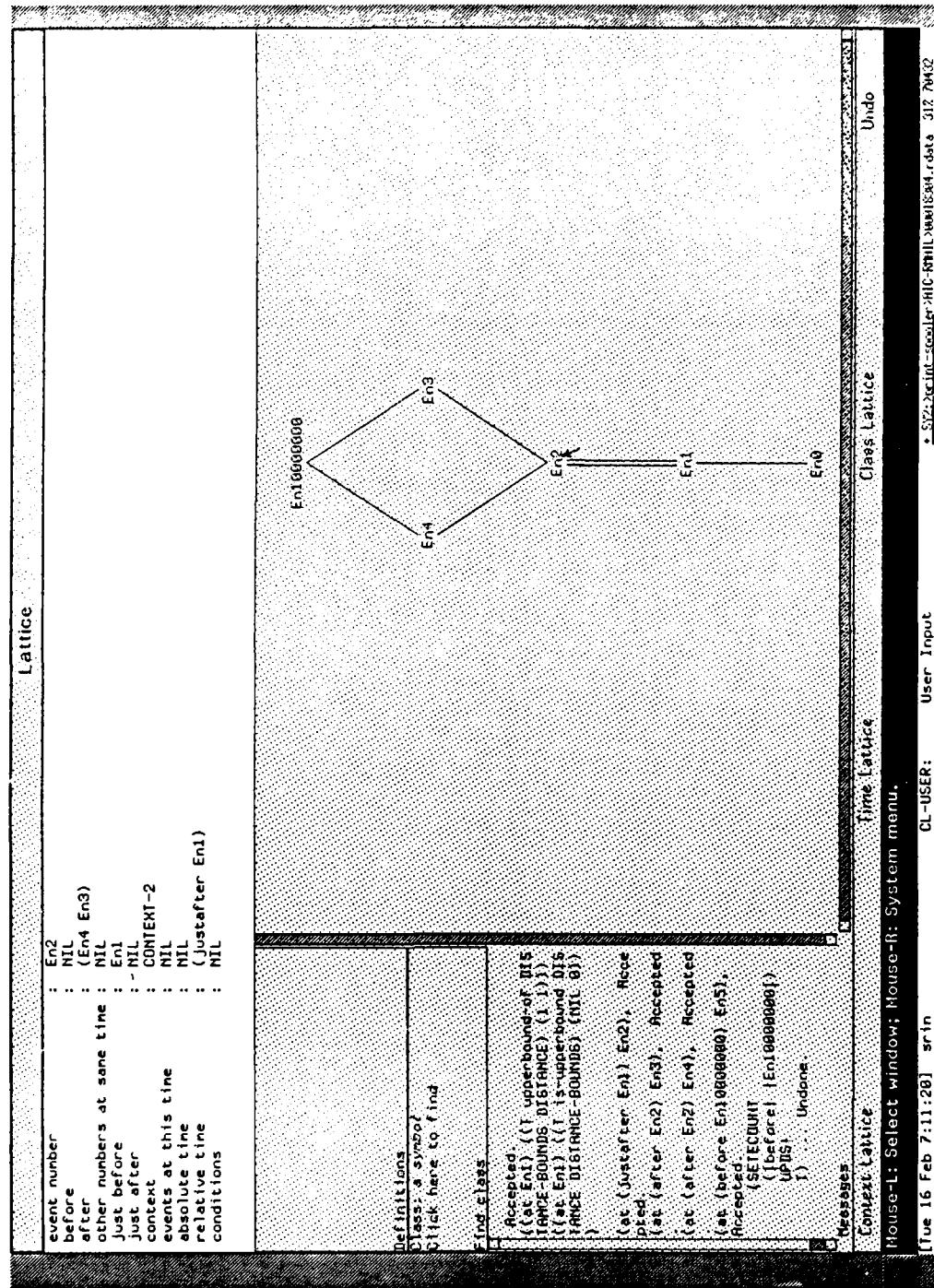
En3

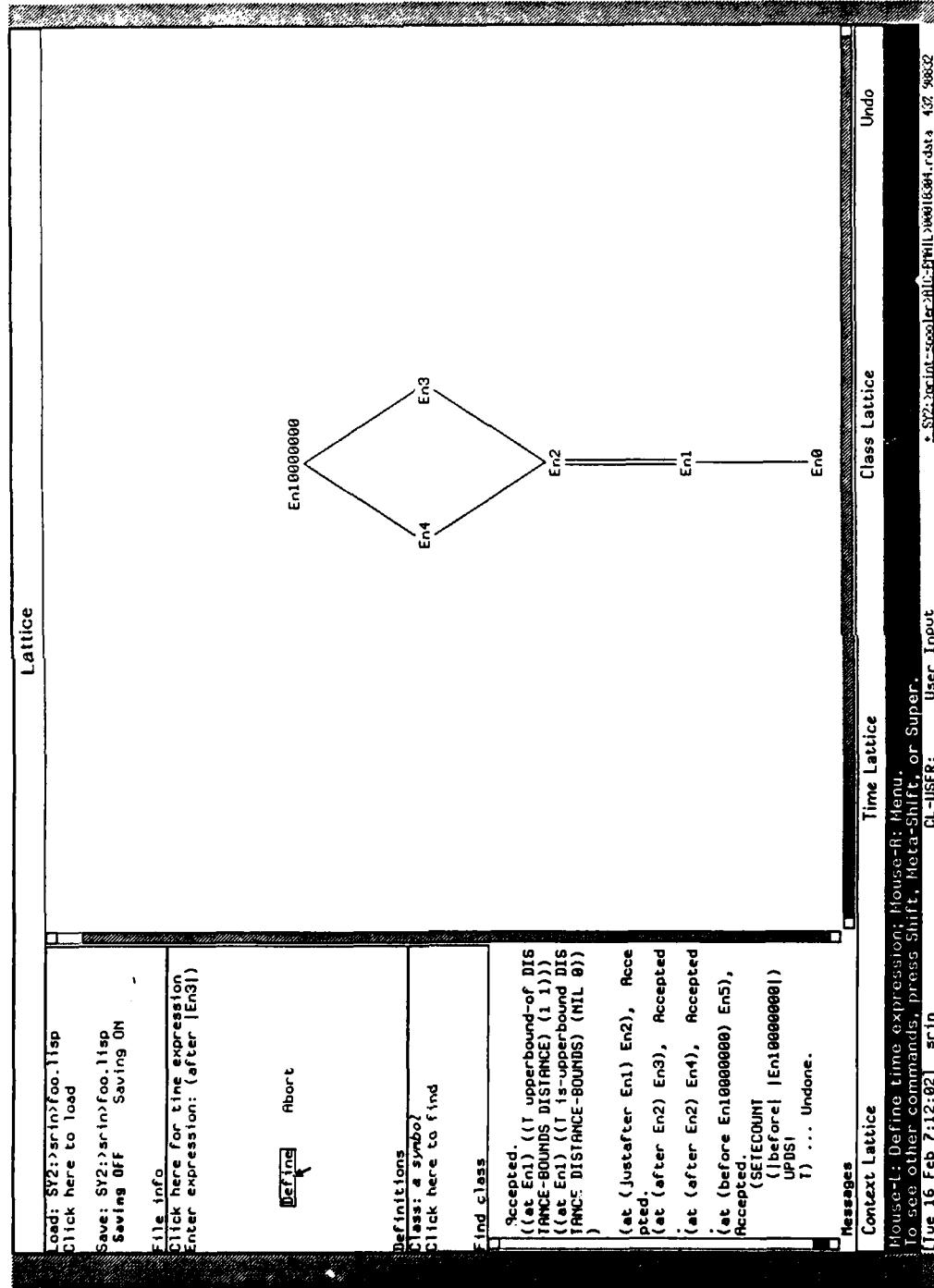
En4

En2

En1

En0





Class Lattice

When the user clicks on the *Class Lattice* command, nodes representing the defined classes are drawn on the display pane with lines showing their relationships to each other. (Figure 8) For purposes of clarity, class names are assigned to node numbers, such as N1, N2, etc. A legend relating the node numbers to the class names is drawn on the right side of the screen. Classes are drawn in a top-down hierarchy, with the most general classes at the top while increasingly specialized classes are drawn below. Since the Class Lattice can get very long, only two levels of classes are shown at a time. Nodes within the same level are staggered to facilitate visibility.

Clicking Left

Placing the mouse over a node name causes appropriate documentation to appear at the bottom of the screen. Clicking on the left mouse button redraws the Class Lattice with the moused node as the root of the tree, allowing two levels beneath it to be drawn. (Figure 9) When a new node becomes the root, the previous root's name is displayed at the top of the legend in bold. Clicking on this name returns the display to its previous configuration. Selecting successive new roots and returning to old ones is one way to browse through the Class Lattice. The other way is by using the Find Class command, about which more later.

Clicking Middle

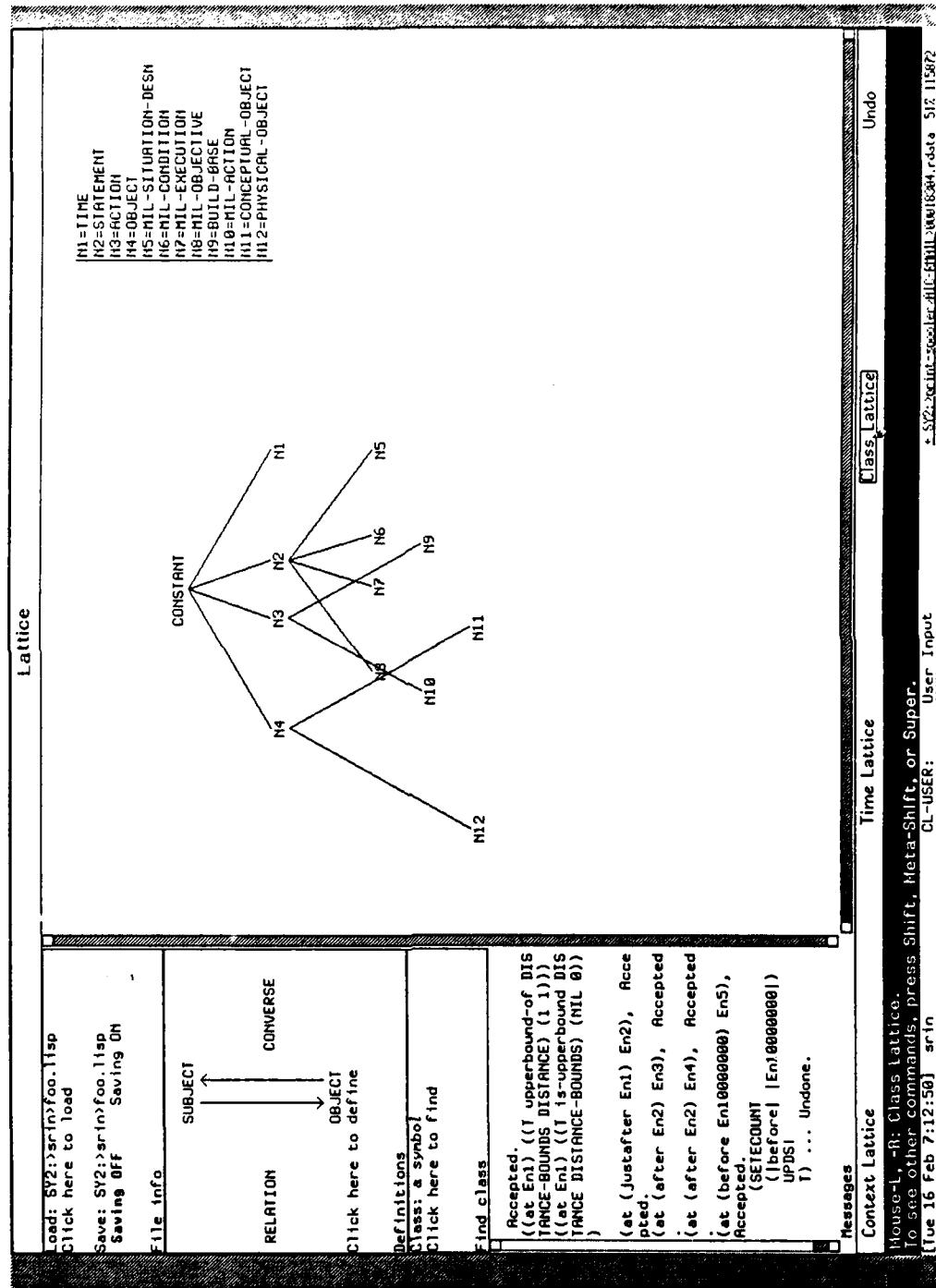
Clicking on the middle mouse button causes a menu to pop up at the location of the mouse offering four choices: *Show dimensions*, *Show inherited dimensions*, *Show immediate instances*, and *Show adopted instances*. (Figure 10) *Show dimensions* causes a window to pop up at the top of the screen, listing all the dimensions defined for the class associated with the moused node. *Show inherited dimensions* lists all the dimensions for a class as well as all the dimensions which it has inherited from its generalizations, separating the two with a horizontal line. (Figure 11) If there are no dimensions, then a message will indicate this in the message pane.

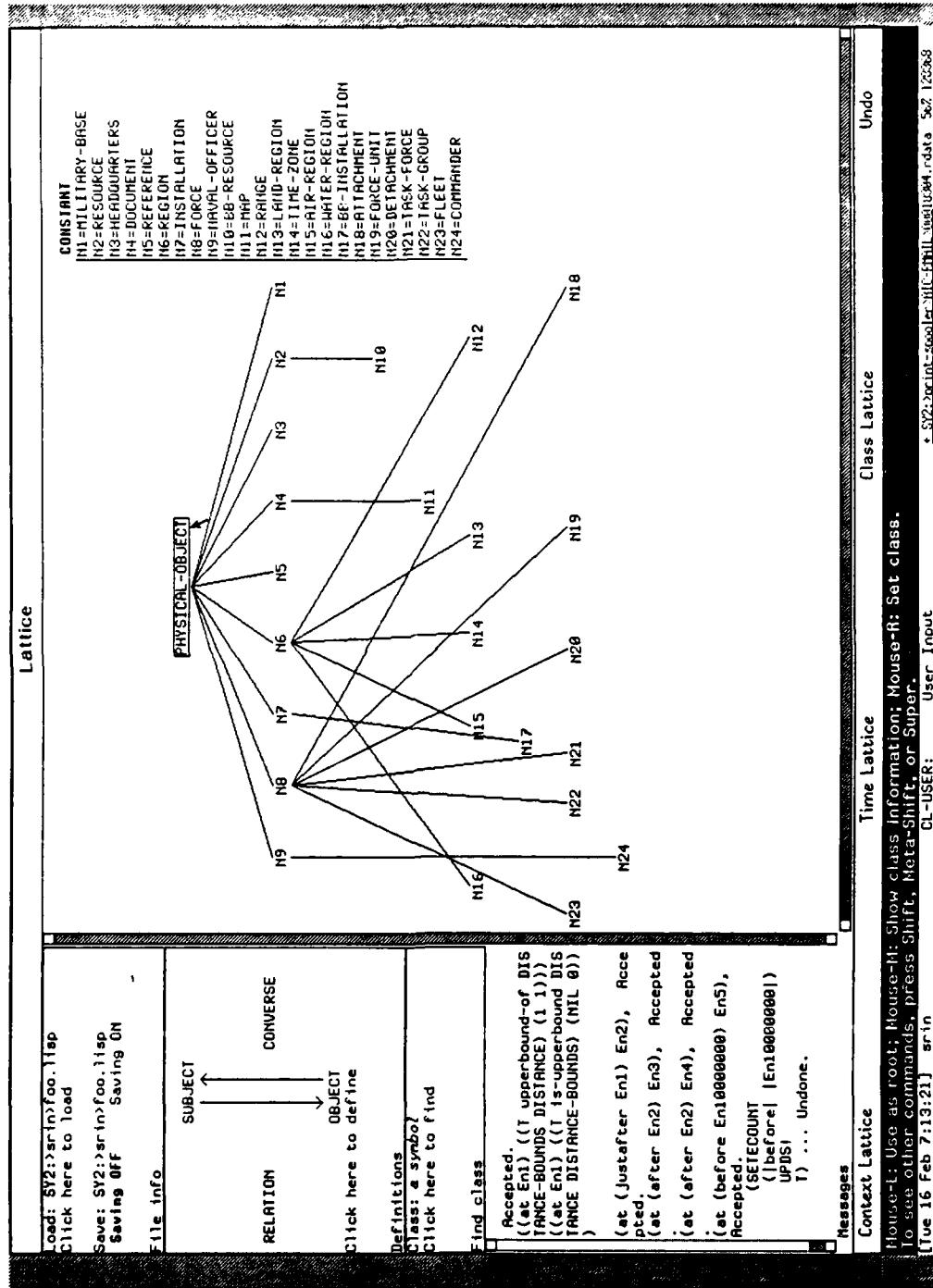
Show immediate instances pops up a menu at the location of the mouse, listing all the instances defined for the class associated with the moused node. Clicking the mouse on one of these instances causes a window to pop up at the top of the screen, listing all the relations defined for this particular instance. *Show adopted instances* pops up a menu listing all the instances of a class as well as all the instances of its specializations, separating the two with a horizontal line. (Figure 12) Clicking the mouse on one of these instances also lists the relations defined for it. (Figure 13) If there are no instances or no relations, then a message will indicate this in the message pane.

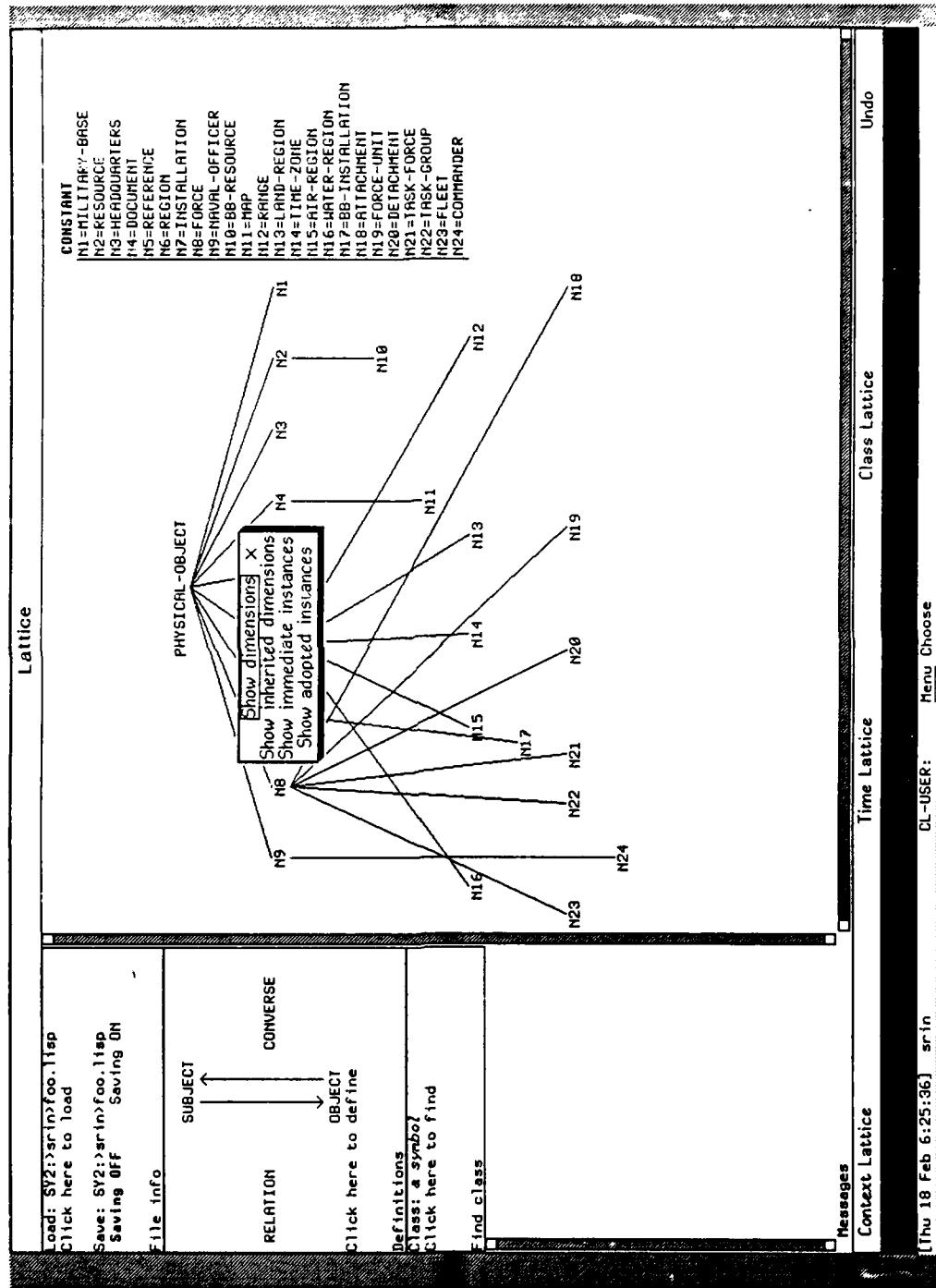
Clicking Right

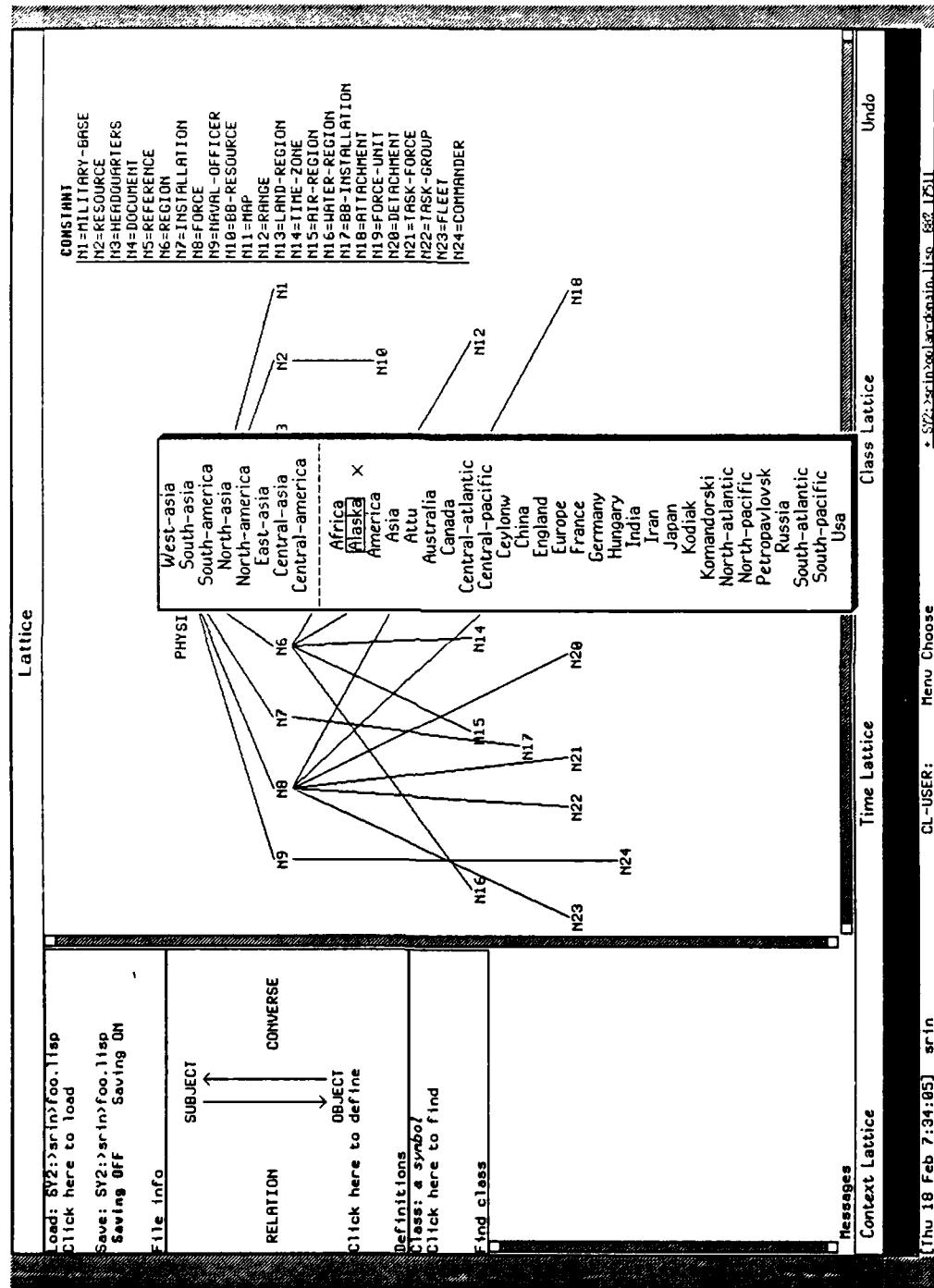
Clicking on the right mouse button pops up a menu at the location of the mouse offering two choices: *Set specializations* and *Create instances*. (Figure 14) *Set specializations* creates new specializations of the class associated with the moused node. A prompting window pops up asking the user for a list of specializations of the given class, a list of additional generalizations besides the given class (the default is no additional classes), and an event number to associate with this new piece of information being entered into the world (the default is the current event number). (Figure 15) Once this entry is completed, the class lattice is redrawn to reflect the addition to the hierarchy.

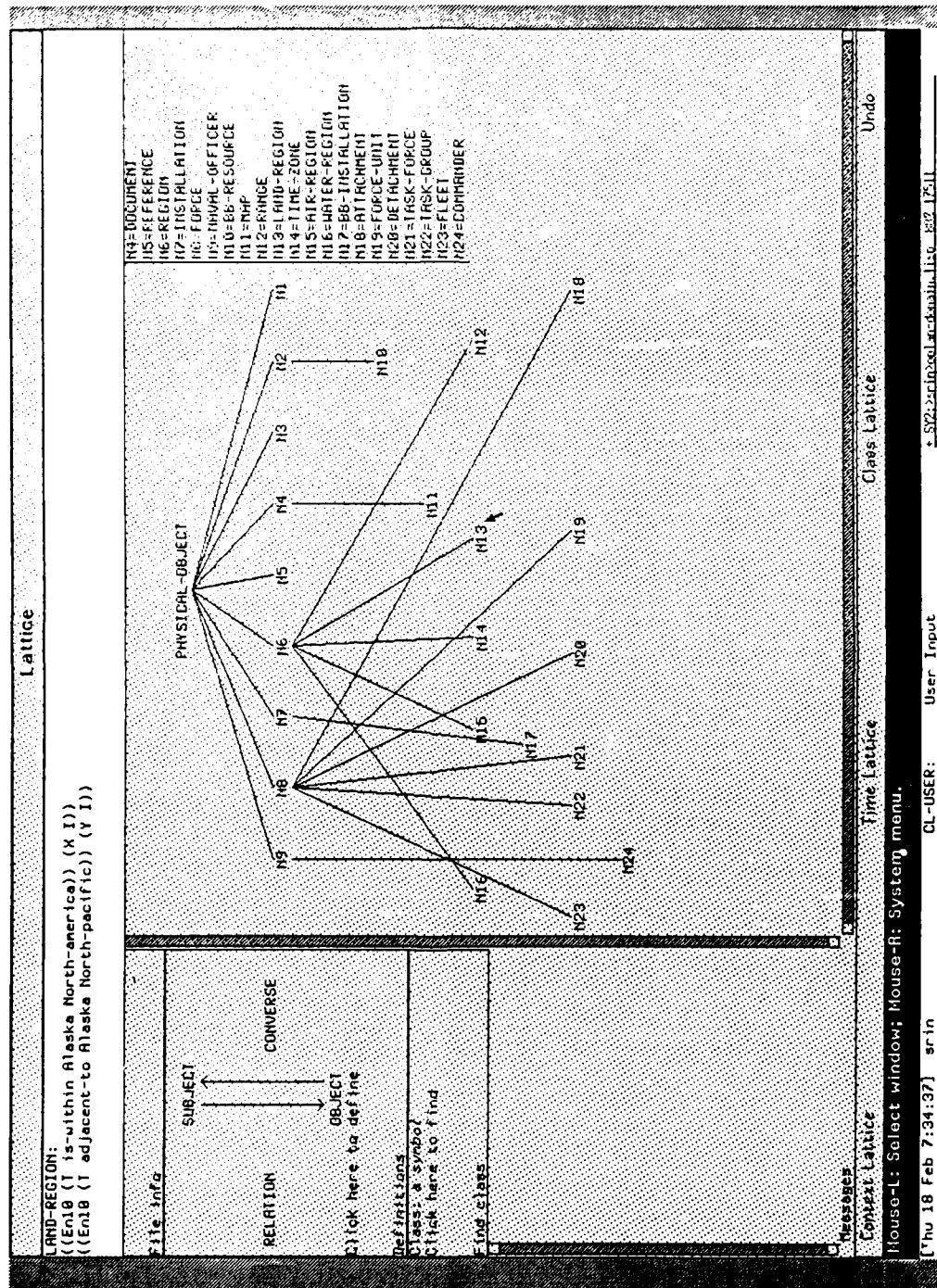
Create instances behaves in a similar manner. A prompting window pops up asking the user for a list of instances of the given class, and an event number to associate with this

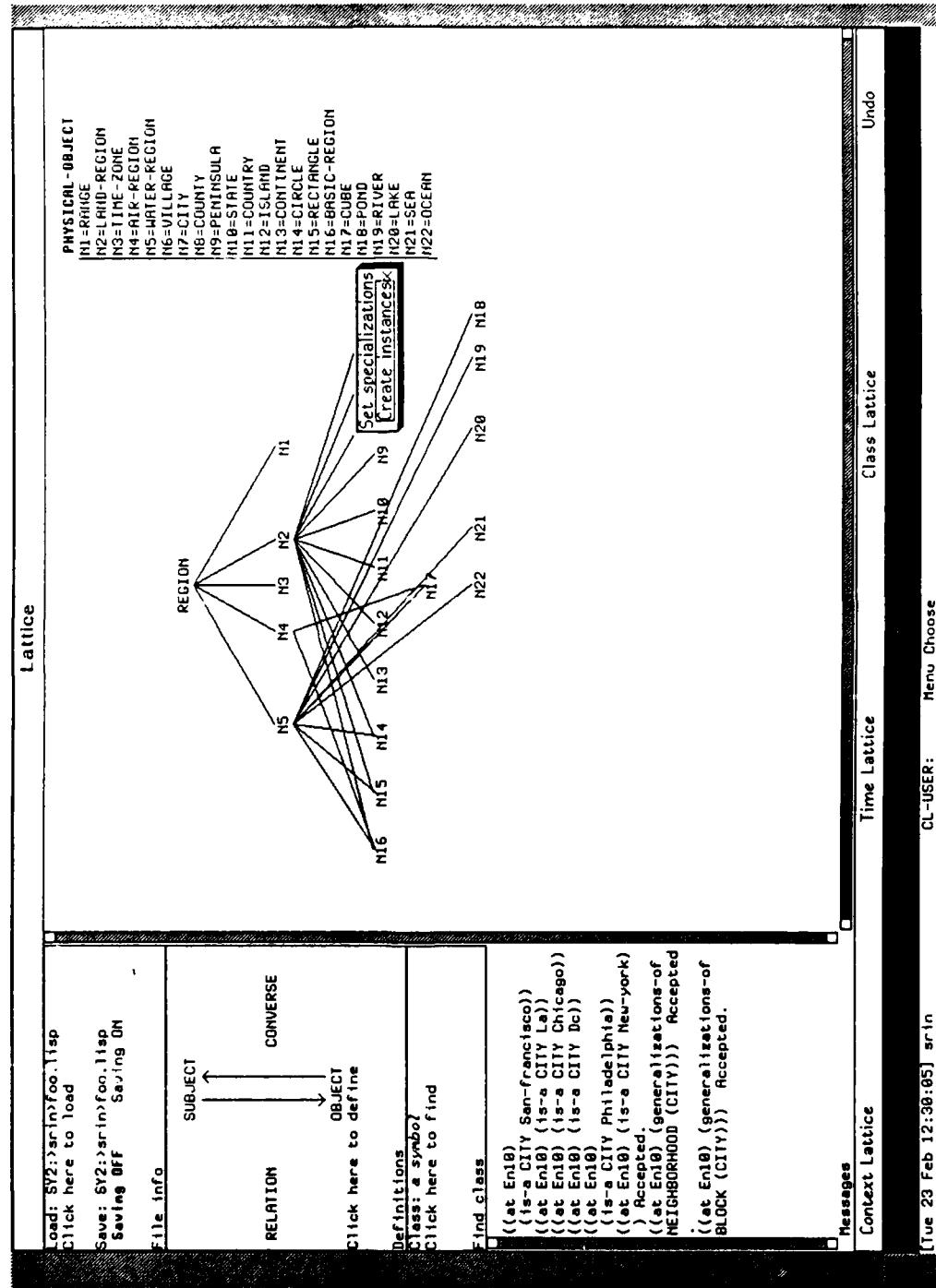


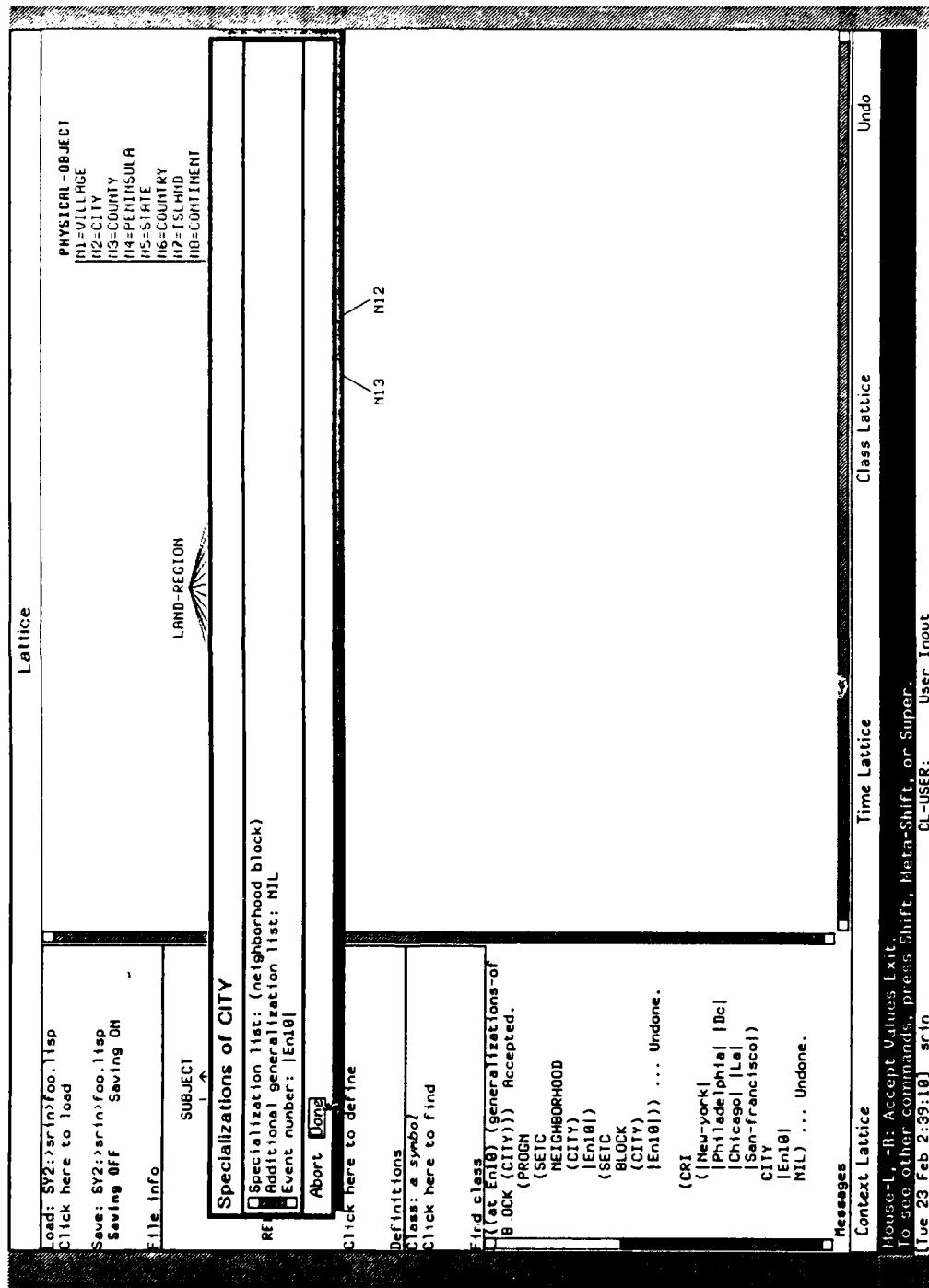












new piece of information (the default is the current event number). (Figure 16) Once this entry has been completed, the new instances can be viewed by clicking on the middle mouse button and selecting either of the *Show instances* commands.

Defining Dimensions

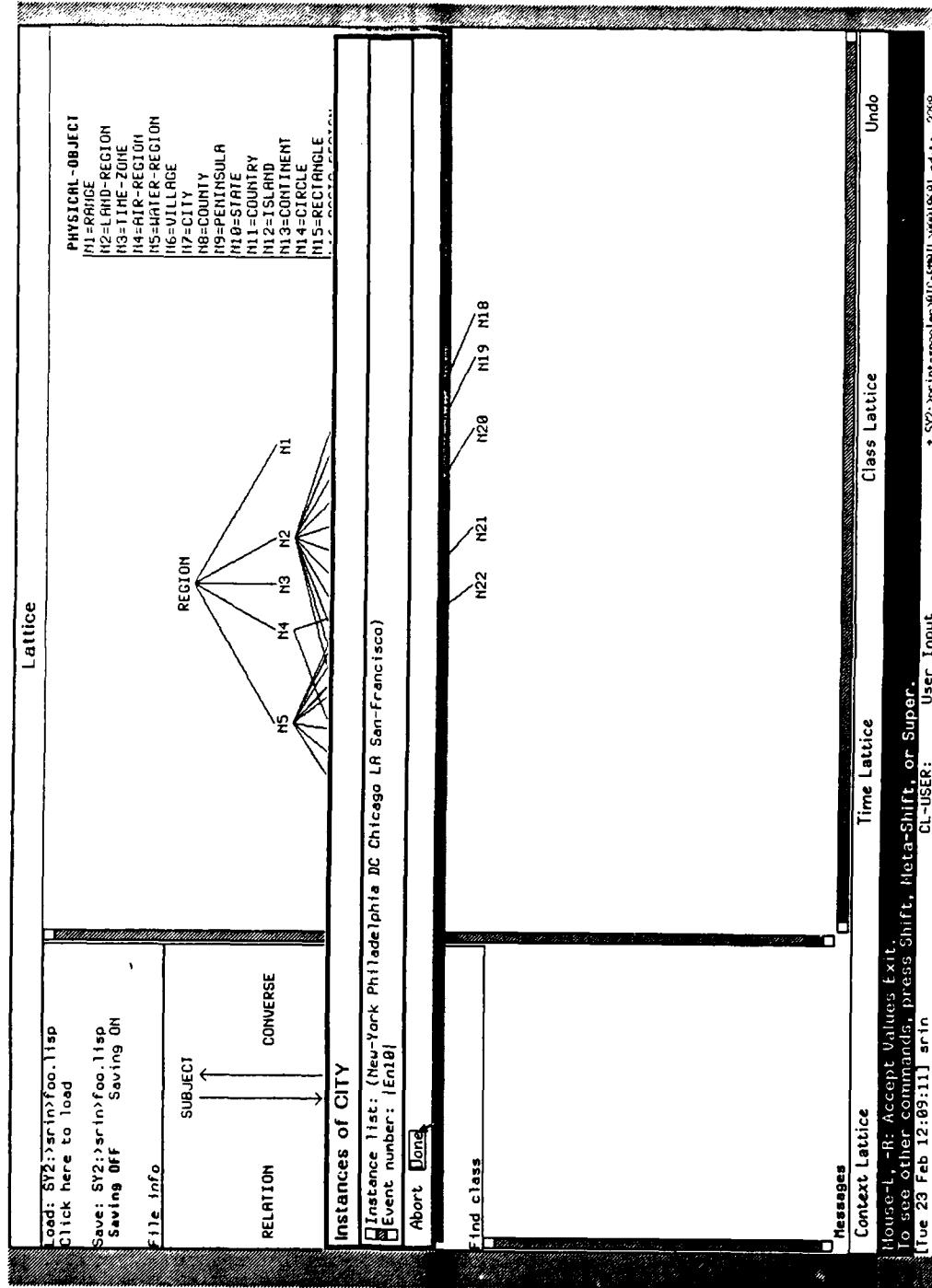
When the Class Lattice is displayed, the Definitions pane provides a map of the relationship between the different components of a dimension. It shows how the subject, given the relation, implies the object, and how the object, given the converse relation, implies the subject. This map is used to create dimensions.

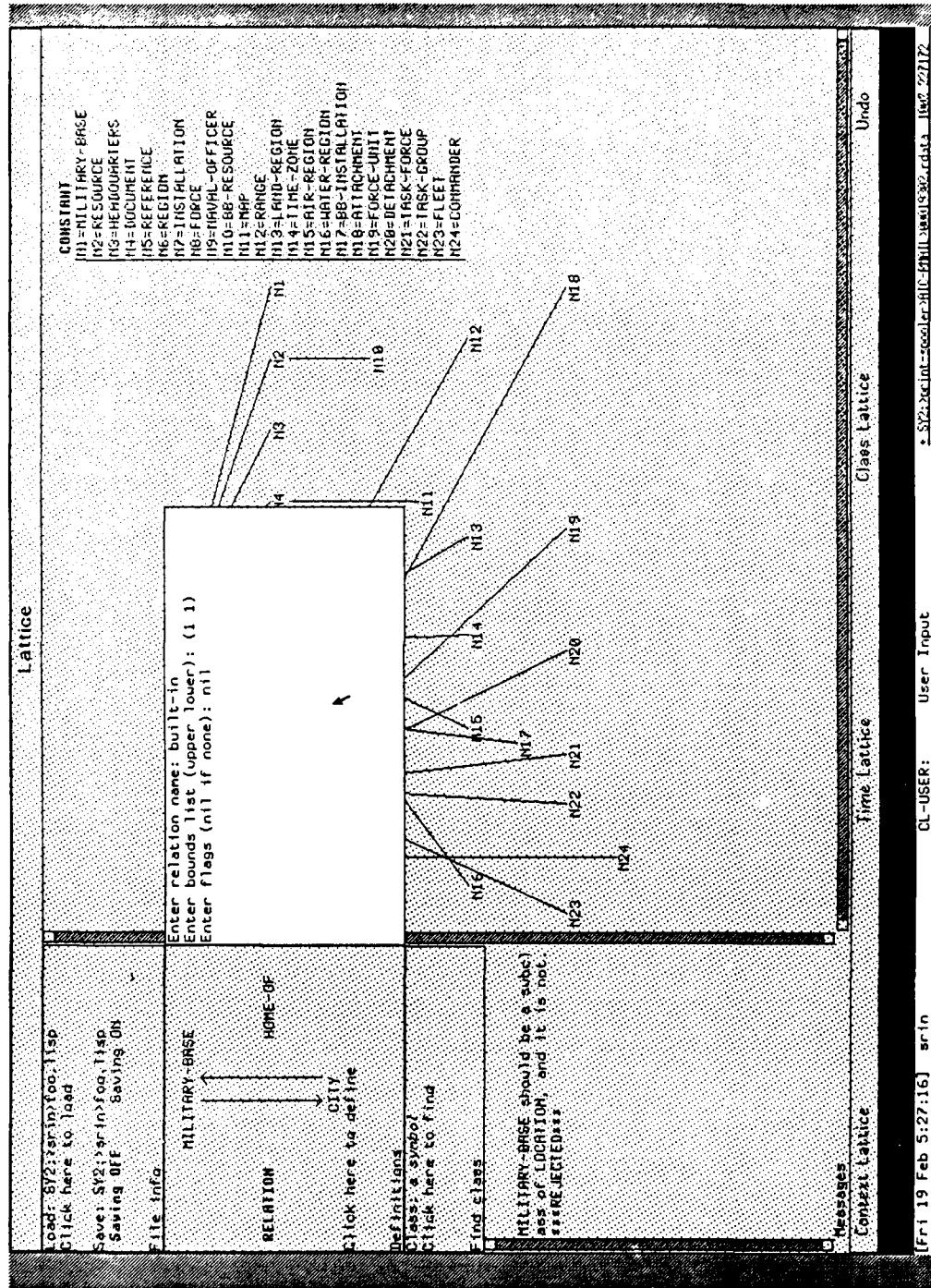
When the user clicks on either the word *SUBJECT* or *OBJECT*, the clicked word turns to bold and the nodes on the lattice go into an argument selection mode. This means that when a node is clicked on, none of the ordinary mouse clicking behavior occurs. Instead, the class represented by that node becomes either the subject or the object of the dimension which is being defined, depending on which the user selected. After the node has been clicked on, the class name of the moused node is displayed in the appropriate position on the map, and the mouse clicking mode returns to normal.

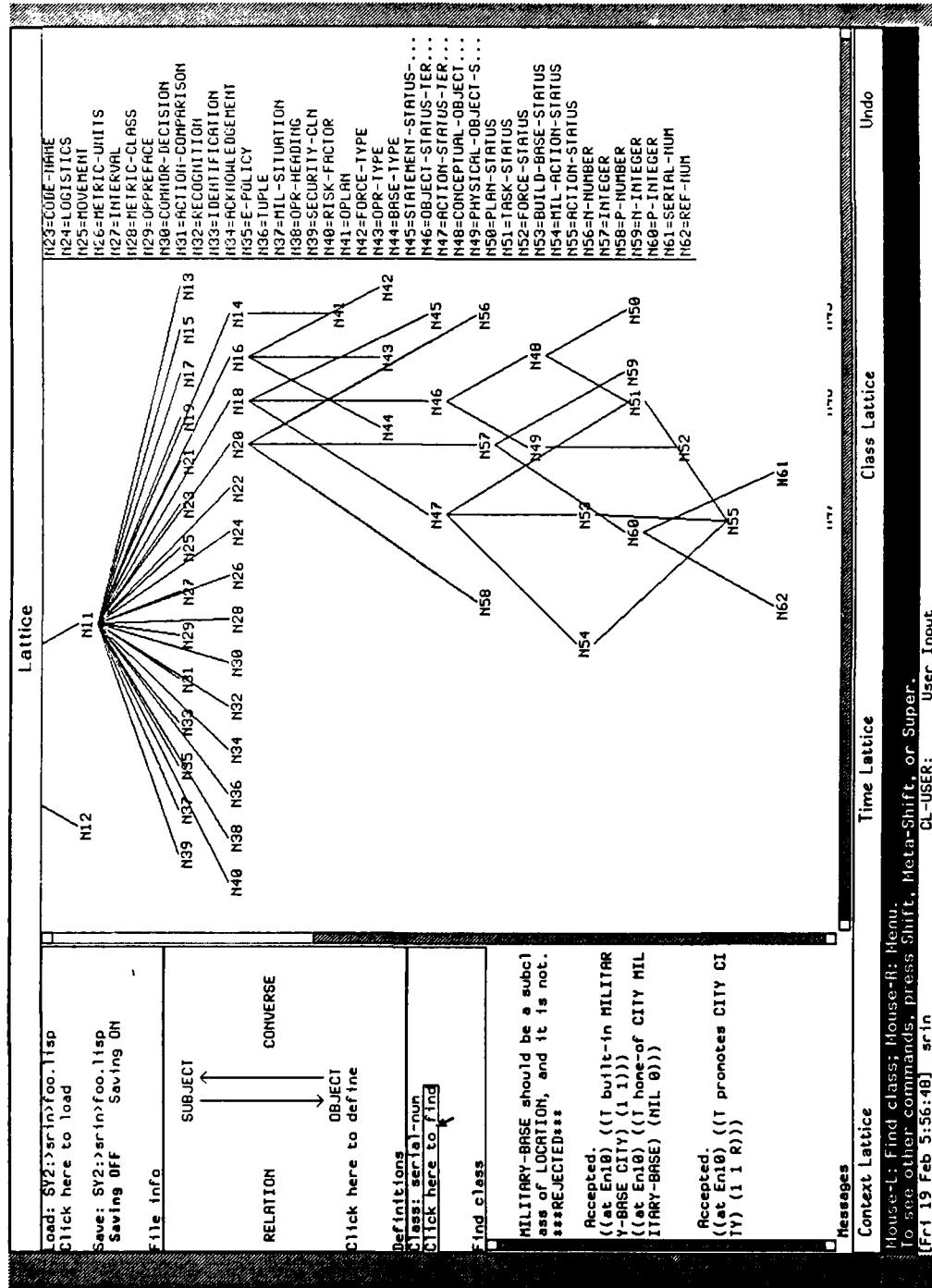
When the user clicks on either the word *RELATION* or *CONVERSE*, a prompting window pops up asking for the name of the relation, a list of the upper and lower bounds, and a list of flags, if any. (Figure 17) Since the relation is a symbol, not a list, it is necessary to type the END key in order to mark the end of input. When defining bounds, *NIL* is used to mean *many* or non-countable. Once this entry is completed, the dimension is defined and the Definitions pane is cleared.

Finding a Class

When the Class Lattice is displayed, the *Find Class* pane is activated. This function provides a fast display of a given class, bypassing the two-level browsing technique. The user enters the name of a class, types the END key, and clicks on the words *Click here to find*. The lattice is drawn in its entirety until it gets to the node of the given class, which is drawn in bold. (Figure 18)







CONCLUSION

The user interface for the CKLOG system enables the user to access and manipulate data representing information in an environment. The environment, or context, can be displayed in relation to other environments that relate to it. New contexts can be created, and the pointer to the context describing the current world state can be moved along the context tree.

The user can also view and manipulate event number information. The event numbers can be displayed showing their relationships to each other, and new event numbers can be entered into the world.

The user can access and perform a variety of operations on class information. All information about a class can be displayed via a series of menu selections. Classes can be added to the hierarchy, dimensions can be set up relating one class to another, and instances of classes can be created. There are two methods for browsing through the Class Lattice - by changing the root of the lattice, or by using the Find Class facility.

Finally, all commands that the user enters can be undone or saved to a file. A file of domain-building commands can be loaded into the system to restore all the information in a domain. Messages are printed on the screen to notify the user of the results of a command.